```
// See https://aka.ms/new-console-template for more information

using System.Security.Cryptography;
using OfficeOpenXml;
using System;
using OfficeOpenXml.Drawing.Chart;
using DocumentFormat.OpenXml.Spreadsheet;
using System.Collections.Generic;
using DocumentFormat.OpenXml.Office2016.Drawing.ChartDrawing;

namespace AnalisisdeseccionV6
{
    class Program
    {
        static void Main(string[] args)
        {
            (List <(double, double, double)> LPMExport, List <(double, double,
double) > LConcExport,
                List <(double, double)> LAceroExport, List<(double, double,
double)>LConcNoConf, string MetAnalisis) = PMM();

            string URL = "D:\\Especialización\\Seminario\\Export";


            Program cExport;
            cExport = new Program();

            cExport.Exportar(LPMExport,LConcExport,LAceroExport, LConcNoConf,
URL, "Análisis en Dir Y" + " 'Condiciones: " + MetAnalisis + "'");


        }


        public void Exportar(List<(double, double, double)> pLPMExport,
List<(double defAs,double esfAs, double i)> pLConcreto,
            List<(double esfAs,double defAs)> pLAcero, List<(double defConcNC,
double fcConcNC, double i)> pLConcNC, string pfilePath, string pNombre)
        {

            //Se crea el archivo y se guarda el nombre para almacenarlo
            Console.WriteLine("Escriba el nombre para guardar del archivo");

            string filename = Console.ReadLine();
            string filename1 = filename + ".xlsx";
            string filepath = Path.Combine(pfilePath, filename1);
            double cantDatos = pLPMExport.Count + 2;

            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

            FileInfo fileInfo = new FileInfo(pfilePath);


            //Se organiza la hoja de cálculo para almacenar los datos del
```

diagrama de interacción

```
            using (ExcelPackage package = new ExcelPackage(fileInfo))
            {

                ExcelWorksheet worksheet =
package.Workbook.Worksheets.Add("Diagrama Interacción");

                int row = 3;

                worksheet.Column(1).Width = 18;
                worksheet.Column(2).Width = 16.5;
                worksheet.Column(3).Width = 16.5;

                worksheet.Column(1).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                worksheet.Column(2).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                worksheet.Column(3).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;


                worksheet.Cells["A1:C2"].Style.Font.Bold = true;
                worksheet.Cells["A1:C1"].Merge = true;

                worksheet.Cells["A1"].Value = pNombre;
                worksheet.Cells["A2"].Value = "Loc eje neutro [mm]";
                worksheet.Cells["B2"].Value = "Carga Axial [kN]";
                worksheet.Cells["C2"].Value = "Momento [kN*m]";

                foreach (var valor in pLPMExport)
                {

                    worksheet.Cells["A" + row].Value = Math.Round(valor.Item1,
2);
                    worksheet.Cells["B" + row].Value = Math.Round(valor.Item2,
2);
                    worksheet.Cells["C" + row].Value = Math.Round(valor.Item3,
2);


                    row++;
                }

                string titulograph = "P vs M " + pNombre;

                var chart = worksheet.Drawings.AddChart(titulograph,
eChartType.XYScatterSmoothNoMarkers);
                chart.SetPosition(2, 0, 4, 0);
                chart.SetSize(600, 400);
                chart.YAxis.Title.Rotation = 270;
                chart.Title.Text = "P vs M";
                chart.XAxis.Title.Text = "Momento [kN*m]";
                chart.YAxis.Title.Text = "Carga Axial [kN]";
```

```
                var serie = chart.Series.Add(worksheet.Cells["B3:B" +
cantDatos], worksheet.Cells["C3:C" + cantDatos]);

                ExcelWorksheet WSPropMat =
package.Workbook.Worksheets.Add("Propiedades Concreto");

                ExcelWorksheet seleccionarhoja =
package.Workbook.Worksheets["Propiedades Concreto"];


                seleccionarhoja.Column(1).Width = 30;
                seleccionarhoja.Column(2).Width = 18;
                seleccionarhoja.Column(3).Width = 5;
                seleccionarhoja.Column(4).Width = 30;

                seleccionarhoja.Column(1).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                seleccionarhoja.Column(2).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                seleccionarhoja.Column(3).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                seleccionarhoja.Column(4).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

                seleccionarhoja.Cells["A1:B1"].Merge = true;

                seleccionarhoja.Cells["A1"].Value = "Esfuerzo - Deformación del
Concreto Confinado";
                seleccionarhoja.Cells["A2"].Value = "Def del Concreto [mm/mm]";
                seleccionarhoja.Cells["B2"].Value = "Fc [MPa]";

                seleccionarhoja.Cells["D1:E1"].Merge = true;

                seleccionarhoja.Cells["D1"].Value = "Esfuerzo - Deformación del
Concreto NC";
                seleccionarhoja.Cells["D2"].Value = "Def del Concreto [mm/mm]";
                seleccionarhoja.Cells["E2"].Value = "Fc [MPa]";


                seleccionarhoja.Cells["A1:E2"].Style.Font.Bold = true;


                row = 3;

                foreach (var valor in pLConcreto)
                {

                    seleccionarhoja.Cells["A" + row].Value =
Math.Round(valor.defAs, 5);
                    seleccionarhoja.Cells["B" + row].Value =
Math.Round(valor.esfAs,2);
                    //seleccionarhoja.Cells["C" + row].Value =
Math.Round(valor.i, 2);
```

```
                    row++;

                }

                row = 3;

                foreach (var valor1 in pLConcNC)
                {

                    seleccionarhoja.Cells["D" + row].Value =
Math.Round(valor1.defConcNC, 5);
                    seleccionarhoja.Cells["E" + row].Value =
Math.Round(valor1.fcConcNC, 2);

                    row++;
                }

                double cant2aux = pLConcNC.Count;

                double cant2 = pLConcreto.Count;
                string titulograph2 = "Esfuerzo - Deformación Concreto";

                var chart2 = seleccionarhoja.Drawings.AddChart(titulograph2,
eChartType.XYScatterSmoothNoMarkers);

                chart2.SetPosition(2, 0, 4, 0);
                chart2.SetSize(600, 400);
                chart2.Title.Text = titulograph2;
                chart2.YAxis.Title.Rotation = 270;
                chart2.XAxis.Title.Text = "Deformación [mm]";
                chart2.YAxis.Title.Text = "Esfuerzo [MPa]";

                chart2.Series.Add(seleccionarhoja.Cells["B3:B" + cant2],
seleccionarhoja.Cells["A3:A" + cant2]);
                chart2.Series.Add(seleccionarhoja.Cells["E3:E" + (cant2aux+2)],
seleccionarhoja.Cells["D3:D" + (2+ cant2aux)]);


                ExcelWorksheet WSPropAce =
package.Workbook.Worksheets.Add("Propiedades Acero");

                ExcelWorksheet HojaAcero =
package.Workbook.Worksheets["Propiedades Acero"];


                HojaAcero.Column(1).Width = 30;
                HojaAcero.Column(2).Width = 18;
                HojaAcero.Column(3).Width = 18;

                HojaAcero.Cells["A1:B1"].Merge = true;

                HojaAcero.Column(1).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                HojaAcero.Column(2).Style.HorizontalAlignment =
```

```
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
                HojaAcero.Column(3).Style.HorizontalAlignment =
OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

                HojaAcero.Cells["A1:B1"].Merge = true;


                HojaAcero.Cells["A1"].Value = "Esfuerzo - Deformación del
Acero";
                HojaAcero.Cells["A2"].Value = "Def del acero [mm/mm]";
                HojaAcero.Cells["B2"].Value = "Fs [MPa]";


                HojaAcero.Cells["A1:C2"].Style.Font.Bold = true;


                row = 3;

                foreach (var valor in pLAcero)
                {

                    HojaAcero.Cells["A" + row].Value = Math.Round(valor.defAs,
5);
                    HojaAcero.Cells["B" + row].Value = Math.Round(valor.esfAs,
2);
                    //seleccionarhoja.Cells["C" + row].Value =
Math.Round(valor.i, 2);

                    row++;

                }

                double cant3 = pLAcero.Count;
                string titulograph3 = "Esfuerzo - Deformación Acero";

                var chart3 = HojaAcero.Drawings.AddChart(titulograph3,
eChartType.XYScatterSmoothNoMarkers);

                chart3.SetPosition(2, 0, 4, 0);
                chart3.SetSize(600, 400);
                chart3.Title.Text = titulograph3;
                chart3.YAxis.Title.Rotation = 270;
                chart3.XAxis.Title.Text = "Deformación [mm]";
                chart3.YAxis.Title.Text = "Esfuerzo [MPa]";

                chart3.Series.Add(HojaAcero.Cells["B3:B" + cant3],
HojaAcero.Cells["A3:A" + cant3]);

                package.SaveAs(new FileInfo(filepath));
            }



        }
```

```
        static  (List<(double, double, double)>, List<(double, double, double)>,
List<(double, double)>, List<(double, double, double)>,string) PMM()
        {
            //Definición de las clases a usar dentro del Método

            Program Cprogram;
            Cprogram = new Program();

            Acero Cacero;
            Cacero = new Acero();

            Concreto Cconcreto;
            Cconcreto = new Concreto();


            // Definición de los modelos constitutivos de los materiales a
utilizar

            string MetodoAcero = "King"; // King ó Bilineal
            string MetodoConcreto = "Mander"; // Whitney ó Mander

            // Dirección de analisis
            string direc = "Y"; //Y ó X (En direc. || Lw ó || a tw
respectivamente)

            // Definición de variables geométricas
            double Lw = 800; //mm Altura del elemento
            double tw = 400; //mm Ancho del elemento
            double r = 40;  //mm  recubrimento

            // Selección del refuerzo longitudinal

            int barl = 6; // Cant. bar. paralelas a lw en una cara
            int barb = 3; // Cant. bar. paralelas a tw en una cara

            double refesq = 510; //mm2 ; área del ref. para las esquinas
            double refr = 284; //mm2 ; área de ref. para el resto

            // Configuración del refuerzo transversal (estribos)

            double Dest = 9.5;//Diámetro del estribo cerrado
            double Sest = 100;//Separación a lo largo del elemento entre línea
de estribos
            double Aest = Math.PI * Dest * Dest / 4; //mm2: Área del estribo

            int nrl = 3; // Número de ramas que van paralelas a Lw (Mínimo 2
ramas)
            int nrb = 6; //Número de ramas que van paralelas a tw ( Mínimo 2
ramas)

            // Definición propiedades de los materiales
            double fc = 28;  //MPa: Resistencia a la compresión del cilindro de
```

```
concreto
            double B1 = Cconcreto.betaW(fc); // Cálculo del beta mediante
Whitney

            double fy = 420; //MPa: Esfuerzo de fluencia para el acero
longitudinal
            double fyh = 420; //MPa: Esfuerzo de fluencia para el acero de
confinamiento (estribos)
            double fsu = 630; //MPa : Esfuerzo último del acero

            double Es = 200000; //MPa Módulo de elásticidad del acero
            double Ec = 4700 * Math.Sqrt(fc); //Módulo de elásticidad para el
concreto

            double ey = 0.0021; //Deformación unit. del acero en fluencia
            double esh = 0.008; //Deformación unit. del acereo al final de la
meceta
            double esu = 0.12; //Deformación última del acero

            double ecu = 0.003; //Deformación última para el concreto no
confinado



            //----------------------------------------Fin de los datos de
entrada-----------------------------------------------//


            // Cálculo de las propiedades de área
            double Agw = Lw * tw; //mm2: Área bruta de la sección
            double Iw = tw * (Math.Pow(Lw, 3)) / 12;//mm2:  Inercia bruta de la
sección

            // Refuerzo para el análisis
            int nbar = 2 * barl + 2 * (barb - 2); //Total de barras en la
sección
            double Ast = (4 * refesq + (nbar - 4) * refr); //mm2: Área de acero
long. total
            double pt = Ast / Agw; // Cuantía de acero long. total

            // Cálculo del área y diámetro promedio
            double Asprom = (4 * refesq + (nbar - 4) * refr) / nbar;//mm2: Área
de acero promedio
            double Dprom = Math.Sqrt(4 * Asprom / Math.PI); //mm; Diámetro
promedio para el refuerzo longitudinal

            double Asprom1 = (2 * refesq + (barb - 2) * refr) / barb; //mm2:
Área promedio de la primera y última fila
            double Dprom1 = Math.Sqrt(4 * Asprom1 / Math.PI); //mm: diámetro
promedio para la primera y última fila del refuerzo

            //Cálculo de la separación del refuerzo
            double Sl = (Lw - 2 * r - Dest * 2 - Dprom1) / (barl - 1);
//Separación del ref. || a Lw
```

```
            double Sb = (tw - 2 * r - Dest * 2 - Dprom1) / (barb - 1); //
Separación del ref. || a tw

            //Crea una lista que almacena: #Fila, Área de ref. / fila y la pos.
resp. a la fib. a comp.
            List<(double, double, double)> Lref = new List<(double, double,
double)>();

            for (int i = 0; i < barl; i++)
            {
                double posicion = r + Dest + Dprom1 / 2;//Posición de la primera
fila de refuerzo

                if (i == 0 || i == barl - 1)
                    Lref.Add((i + 1, 2 * refesq + (barb - 2) * refr, posicion +
i * Sl));
                else
                    Lref.Add((i + 1, 2 * refr, posicion + i * Sl));
            }
            foreach (var varref in Lref)
                Console.WriteLine($"fila: {varref.Item1}, Área ref:
{varref.Item2}mm2,Posición: {varref.Item3}mm");


            // Definición de los puntos para el diagrama PM
            int numpuntos = 50; //Cantidad de puntos en el diagrama PM
            double tp = Lw / numpuntos; // Tamaño de paso
            double c; //mm; Posición del eje neutro
            double esi;
            double fsi;
            double Csi;
            double Msi;

            ////////////////////

            double Mt = 0; //Momento total para un "c" dado
            double Pt; //Carga axial total para un "c" dado

            ////////////////////// Definición de las variables para concreto
confinado
            double Gamma; // Posición de la resultante de los esfuerzos en el
concreto
            double Alfa;  // Conf. del Concreto ó Factor de área equivalente
            double fcc; // Esfuerzo para el concreto confinado
            double eccu; // Deformación última para el concreto confinado
            ////////////////// Variables para concreto no confinado
            double Gamma2;
            double Alfa2;
            double fc2;
            List<(double, double, double)> LMandernoconfinado = new
List<(double, double, double)>(); //Lista pra. alm. esf vs def de mander no
confinado
            List<(double, double)> LEsDefAs = new List<(double, double)>();
//Lista para al esf vs def del acero
```

```
            double cu1;
            double cu2;

            // Definición de la carga axial máxima

            double fccP = 0;
            double AlfaP = 0;
            double eccuP = 0;
            double Ps = 0;
            double fsP = 0;
            double Anc = (tw - 2 * r) * (Lw - 2 * r);
            double Pmax = 0;

            (Gamma, AlfaP, fccP, eccuP, List<(double, double, double)>
LesConcreto) = Cconcreto.MMander(fc, fyh, Ast, Aest, Dprom, r, Dest, Lw, tw,
Sest, nrb, nrl, Ec, esu, direc);

            // Componente de tracción pura en el acero según el método selecc.

            if (MetodoAcero == "King")
            {
                (fsP,LEsDefAs) = Cacero.Mking(eccuP, fy, ey, Es, esh, esu, fsu);
                Ps = Ast * fsP / 1000;
            }
            else
                Ps = Ast * fy / 1000;

            // Cálculo de la compresión máxima en el concreto según el método
selecc.
            if (MetodoConcreto == "Whitney")
            {
                //Pt = 0.75 * (0.85 * fc * (Agw - Ast)) / 1000;
                Pmax = 0.75 * (0.85 * fc * (Agw - Ast) + Ps * 1000) / 1000;
            }
            else
                Pmax = 0.75 * (AlfaP * fccP * (Anc - Ast) + 0.85 * fc * (Agw -
Anc) + Ps * 1000) / 1000;

            ////////////////////////////////

            List<(double, double, double)> Les = new List<(double, double,
double)>(); //Lista pra. alm. def. en el acero.
            List<(double, double, double)> Lfzs = new List<(double, double,
double)>(); //Lista pra. alm. las fzs y Momentos. en el acero.
            List<(double, double, double)> LMvsP = new List<(double, double,
double)>(); //Lista que almacena los pares de P y M


            if (MetodoAcero == "King")
                Pt = -Ast * fsu / 1000;
            else
                Pt = -Ast * fy / 1000;
```

```
            LMvsP.Add((0, Pt, Mt)); //Se agrega el primer punto del diagrama
Mt=0 Pt la máxima tracción

            for (c = tp; c <= Lw;)
            {
                for (int filai = 0; filai < barl; filai++)
                {
                    double di = Lref[filai].Item3;
                    double Asi = Lref[filai].Item2;

                    esi = ecu * (c - di) / c;

                    if (MetodoAcero == "King")
                        (fsi,LEsDefAs) = Cacero.Mking(esi, fy, ey, Es, esh, esu,
fsu);//MPa
                    else
                        (fsi, LEsDefAs) = Cacero.Mbilineal(esi, fy, ey, Es,
esu);

                    Csi = fsi * Asi / 1000; //kN
                    Msi = Csi * (Lw / 2000 - di / 1000);//kN.m
                    Les.Add((filai + 1, esi, fsi));// Almacenamiento de datos
fila, def. Unit. y Esf en el acero
                    Lfzs.Add((filai + 1, Csi, Msi));// Almacenamiento de datos
fila, Fza acero y Momento acero
                }

                //Ciclo que permite obtener los aportes de P (kN) y M (kNm) para
el refuerzo

                //foreach (var def1 in Les)
                //  Console.WriteLine($"fila: {def1.Item1}, Def. Unit. (es):
{def1.Item2},fs: {def1.Item3}MPa");

                //foreach (var fza1 in Lfzs)
                //  Console.WriteLine($"fila: {fza1.Item1}, Tensión (Cs):
{fza1.Item2}kN, Momento (Ms): {fza1.Item3}kN.m");

                double SMs = 0; //Variable que almacena la sumatoria de Momentos
en el acero
                double SFs = 0; //Variable que almacena la sumatoria de fuerzas
en el acero

                //Ciclo para acumular en la variable SFs la sumatoria de fuerzas
en el acero
                foreach (var valorFi in Lfzs)
                    SFs += valorFi.Item2; //Variable para acumular las fuerzas
en el acero
                //Console.WriteLine("Suma de fzas en el acero es: Cs= {0}kN",
SFs);

                //Ciclo para acumular en la variable SMs la sumatoria de
Momentos en el acero
                foreach (var valorMi in Lfzs)
```

```
                    SMs += valorMi.Item3; //Variable para acumular los Momentos
en el acero
                //Console.WriteLine("Suma de Momentos en el acero es: Ms=
{0}kN.m", SMs);

                Les.Clear();
                Lfzs.Clear();

                double Cc;  //Resultante del esfuerzo en el nucleo (Mander)
total en Whitney
                double Ccr1; //Resultante del esfuerzo en el recubrimiento para
la curva
                double Ccr2; //Resultante del esfuerzo en el recubrimiento para
la zona triángular
                double Mci;
                //double dnc; //Prof. del bloque de compresiones para el
recubrimiento

                //Análisis del concreto Confinado o No confinado

                if (MetodoConcreto == "Whitney")
                {
                    // Función para obtener la resultante del bloque de
compresiones
                    Cc = Cconcreto.MWhitney(c, fc, tw); //Resultante del bloque
de compresiones
                    //Console.WriteLine("Resultante del bloque de compresiones:
Cc = {0}kN ", Cc);
                    Mci = Cc * (Lw / 2000 - B1 * c / 2000); //kN.m; Momento
generado por el bloque de compresiones
                }
                else
                {
                    (Gamma, Alfa, fcc, eccu, List<(double, double, double)>
EsfConcreto) = Cconcreto.MMander(fc,
                        fyh, Ast, Aest, Dprom, r, Dest, Lw, tw, Sest, nrb, nrl,
Ec, esu, direc);
                    //Console.WriteLine("Gamma = {0} : Alfa {1} ", Gamma, Alfa);
                    Cc = Alfa * fcc * (tw - 2 * r) * c / 1000;

                    (LMandernoconfinado, Gamma2, Alfa2, fc2) =
Cconcreto.MManderNoConfinado(fc, Ec);
                    cu1 = 0.004 * c / eccu; //Profunidad para la zona curva
                    cu2 = 0.0064 * c / eccu; // Profundidad para todo el
diagrama
                    //dnc = ecu * c / eccu;
                    //Ccr1 = Cconcreto.MWhitney(dnc, fc, 2 * r);
                    //Mci = Cc * (Lw / 2000 - Gamma * c / 1000) + Ccr1 * (Lw /
2000 - B1 * dnc / 2000); //Momento considerando recubrimiento con Whitney
                    Ccr1 = Alfa2 * fc * (2 * r) * cu1 / 1000;
                    Ccr2 = 0.5 * fc2 * (cu2 - cu1) * (2 * r) / 1000;
                    Mci = Cc * (Lw / 2000 - Gamma * c / 1000) + Ccr1 * (Lw /
2000 - Gamma2 * cu1 / 1000) + Ccr2 * (Lw / 2000 - (cu1 + cu2 / 3) / 1000);
//kN.m
```

```
                }

                Mt = Mci + SMs; //Momento total para un "c" dado
                Pt = Cc + SFs; //Carga axial total para un "c" dado

                if (Pt > Pmax)
                    Pt = Pmax;

                //Console.WriteLine("Eje neutro = {2}mm; Momento = {0}kN.m ;
Axial = {1}kN", Mt, Pt,c);
                LMvsP.Add((c, Pt, Mt));

                c = c + tp;

            }


            LMvsP.Add((Lw, Pmax, 0)); //Se agrega el último punto del diagrama
Mt=0 Pt la máxima compresión


            foreach (var MPMP in LMvsP)
                Console.WriteLine("Eje neutro (c) = {0} mm : Momento (M) = {2}
kN.m : Axial (P) = {1} kN", Math.Round(MPMP.Item1, 1), Math.Round(MPMP.Item2,
1), Math.Round(MPMP.Item3, 1));


            Console.WriteLine("F'cc = {0}MPa : eccu={1} : AlfaM = {2} :
Fs(eccu)={3}MPa", fccP, eccuP, AlfaP, fsP);

            //Se crea una lista para almacenar los datos del mander no confinado

            List<(double, double, double)> LManderNoConf = new List<(double,
double, double)>();
            double gamma1 = 0;
            double alfa1 = 0;
            double fcaux = 0;

           (LManderNoConf,gamma1, alfa1, fcaux) =
Cconcreto.MManderNoConfinado(fc, Ec);

            string MetAnalisis = MetodoAcero + " & " + MetodoConcreto;

            return (LMvsP,LesConcreto, LEsDefAs, LManderNoConf, MetAnalisis);
        }

    }

    class Acero
    {
        //Definición del modelo constitutivo Bilineal fs = fy para es>0.0021
        public (double, List<(double, double)>) Mbilineal(double es, double fy,
double ey, double Es, double esu)
        {
```

```
            double es1 = Math.Abs(es);
            double fs = 0;
            double fs1 = 0;
            List<(double esfAce, double defAce)> LEsfDefAce = new List<(double
esfAce, double defAce)>();

            if (es1 <= ey)
                fs = Es * es;

            if (es1 > ey)
                fs = fy * es1 / es;

            for (double j = 0; j <= esu;)
            {
                if (j <= ey)
                    fs1 = Es * j;

                if (j > ey)
                    fs1 = fy * j / j;
                LEsfDefAce.Add((fs1, j));

                j = j + esu / 150;

            }
            return (fs,LEsfDefAce);
        }

        //Definición del modelo constitutivo de "King" para considerar el
endurecimiento por deformación fs>fy para es>0.008
        public (double, List<(double, double)>) Mking(double es, double fy,
double ey, double Es, double esh, double esu, double fsu)
        {
            double es1 = Math.Abs(es); //Valor absoluto de la deformación para
simplificar el método
            double rs = esu - esh;
            double fs = 0;
            double ms = ((fsu / fy) * (Math.Pow((30 * rs + 1), 2)) - 60 * rs -
1) / (15 * Math.Pow(rs, 2));
            double fs1 = 0;

            List<(double esfAs, double defAs)> LEsDef = new List<(double esfAs,
double defAs)>();

            if (es1 <= ey)
                fs = Es * es;

            if (es1 > ey && es1 < esh)
                fs = fy * (es1 / es);
            // Incluye la zona de endurecimiento por deformación
            if (es1 > esh)
                fs = (es / es1) * fy * ((ms * (es1 - esh) + 2) / (60 * (es1 -
esh) + 2) + ((es1 - esh) * (60 - ms)) / (2 * Math.Pow((30 * rs + 1), 2)));
```

```
            for (double j = 0;j <= esu;)
            {
                if (j <= ey)
                    fs1 = Es * j;

                if (j> ey && j < esh)
                    fs1 = fy * (j /j);
                // Incluye la zona de endurecimiento por deformación
                if (j > esh)
                    fs1 = (1) * fy * ((ms * (j - esh) + 2) / (60 * (j - esh) +
2) + ((j - esh) * (60 - ms)) / (2 * Math.Pow((30 * rs + 1), 2)));

                LEsDef.Add((fs1, j));

                j = j + esu / 150;

            }

            return (fs, LEsDef);

        }

    }

    class Concreto
    {
        //------------------------------------------ DEFINICIÓN DEL MÉTODO DE
WHITNEY-------------------------------------//
        public double betaW(double fc)
        {
            double Beta1 = 0.85;
            if (fc < 28)
                Beta1 = 0.85;

            if (fc > 28 && fc < 56)
                Beta1 = (56 - fc) * 0.2 / 28 + 0.65;

            if (fc > 56)
                Beta1 = 0.65;

            return Beta1;
        }

        public double MWhitney(double c, double fc, double tw) //Bloque de
Whitney
        {
            Concreto Cconcreto;
            Cconcreto = new Concreto();

            double Beta1 = Cconcreto.betaW(fc);
            double Alfa = 0.85; //Confiabilidad del concreto
            double Cc = Alfa * fc * tw * c * Beta1 / 1000; //kN: Resultante del
bloque de compresiones
            return Cc;
```

```
        }

        //---------------------------------------- DEFINICIÓN DEL MÉTODO DE
MANDER CONFINADO----------------------------------------//


        //public List<(double, double, double)> MMander(double eci, double fc,
double fyh, double Ast, double Aest, double Dprom, double r, double Dest, double
Lw, double tw, double Sest, int nrb, int nrl, double Ec, double esu, string
direc)
        public (double, double, double, double, List<(double,double,double)>)
MMander(double fc, double fyh, double Ast, double Aest, double Dprom, double r,
double Dest, double Lw, double tw, double Sest, int nrb, int nrl,
            double Ec, double esu, string direc)
        {

            double eco = 0.002; //Deformación unit. del concreto para el f'c del
cilindro
            double bc = tw - 2 * r - Dest; //mm: Base del nucleo al eje del
estribo
            double hc = Lw - 2 * r - Dest; //mm :Altura del nucleo al eje del
estribo
            double s1 = Sest - Dest; //mm: Separación libre entre estribos

            double wlb = (bc - Dprom - Dest) / (nrl - 1) - Dprom;//mm: Sep.
prom. entre bar. soport. por una rama || a la base (tw)
            double wlh = (hc - Dprom - Dest) / (nrb - 1) - Dprom;//mm: Sep.
prom. entre bar. soport. por una rama || a la altura (Lw)

            double Ac = bc * hc; //mm2: Área encerrada por el eje del estribo
            double pcc = Ast / Ac; // Cuantía long. respecto al nucleo confinado

            int Ncurb = 2 + 2 * (nrl - 2); // Cant. de parábol. en sentido || b
            int Ncurh = 2 + 2 * (nrb - 2); // Cant. de parábol. en sentido || h
            int Ncur = Ncurb + Ncurh;// Total de parábolas de conf. efect. en la
sección

            double Aun1 = Ncurb * wlb * wlb / 6 + Ncurh * wlh * wlh / 6;
            double Aun = Math.Min(Ac, Aun1);//mm2: Área sin conf. efectivo
            double Ae = (Ac - Aun) * (1 - (s1 / (2 * bc))) * (1 - (s1 / (2 *
hc)));//mm2: Área conf. efectivamente
            double ke = ((1 - Aun / Ac) * (1 - s1 / (2 * bc)) * (1 - (s1 / (2 *
hc)))) / (1 - pcc);//Coeficiente de confinamiento efectivo.

            double psx = (nrb * Aest) / (hc * Sest); //Cuantía lat. de conf. en
(tw)
            double psy = (nrl * Aest) / (bc * Sest); //Cuantía lat. de conf. en
(Lw)
            double pst = psy + psx;//Cuantía de confinamiento total

            double flb = fyh * psx * ke;//MPa: Esfuerzo lateral de confinamiento
en (tw)
            double flh = fyh * psy * ke;//MPa: Esfuerzo lateral de confinamiento
en (Lw)
```

```
            double fcc = 0; //Def. Var. Resistencia a la comp. Conc. Conf.
            double ecc = 0; //Def. Var. Deformac. unit. para un esf. f'cc
            if (direc == "X")
            {
                fcc = fc * (-1.254 + 2.254 * Math.Sqrt(1 + (7.94 * flb / fc)) -
2 * flb / fc);//MPa:Res. a la comp. del conc. conf. en la direcc. de (b)
                ecc = eco * (1 + 5 * (fcc / fc - 1));//Def. Unit para el
esfuerzo f'cc del Conc. Conf.
            }
            else
            {
                fcc = fc * (-1.254 + 2.254 * Math.Sqrt(1 + (7.94 * flh / fc)) -
2 * flh / fc);//MPa:Res. a la comp. del conc. conf. en la direcc. de (h)
                ecc = eco * (1 + 5 * (fcc / fc - 1));//Def. Unit para el
esfuerzo f'cc del Conc. Conf.
            }

            double Esec = fcc / ecc; //MPa: Mod. Elást. Secante
            double rc = Ec / (Ec - Esec);// Relación de rigideces Esec/Etan

            double eccu = 1.4 * (0.004 + (1.4 * pst * fyh * esu) / fcc);//Def.
Unit. última para concreto confinado

            //Cálculo del esfuerzo medio (alfa) y del centroide para cualquier
def (gamma).

            int Npuntos = 305; // Cantidad de Puntos para la curva de esf. vs
def. conc. conf.
            double xi1 = 0;
            double alfa = 0;
            double gamma = 0;
            double eci1 = 0;
            double fci1 = 0;
            double tp = eccu / Npuntos; //Tamaño de paso
            double areaA = 0;
            double areaG = 0;
            double area_acumuladaA = 0;
            double area_acumuladaG = 0;

            List<(double, double, double)> Lesfmander = new List<(double,
double, double)>(); //Lista pra. alm. esf vs def de mander confinado
            List<(double, double, double)> Lcalalfa = new List<(double, double,
double)>(); // Lista pra. alm. val. del cál. de alfa
            List<(double, double, double)> Lcalgamma = new List<(double, double,
double)>(); // Lista pra. alm. val. del cál. de gamma

            for (double i = 0; i <= Npuntos; i++)
            {
                Lesfmander.Add((eci1, fci1, xi1));
                xi1 = eci1 / ecc;
                fci1 = fcc * xi1 * rc / (rc - 1 + Math.Pow(xi1, rc));
                //Console.WriteLine("eci={0}; fci = {1}; Xi={2}",
eci1,fci1,xi1);
```

```
                eci1 = eci1 + tp;
            }

            for (int filai = 0; filai < Npuntos; filai++)
            {

                areaA = ((Lesfmander[filai + 1].Item1 - Lesfmander[filai].Item1)
* (Lesfmander[filai].Item2 + Lesfmander[filai + 1].Item2)) / 2;
                area_acumuladaA += areaA;
                if (Lesfmander[filai].Item1 == 0)
                    alfa = 0;
                else
                    alfa = area_acumuladaA / (fcc * Lesfmander[filai].Item1);

                Lcalalfa.Add((areaA, area_acumuladaA, alfa));
            }


            for (int filaj = 0; filaj < Npuntos; filaj++)
            {

                areaG = ((Lesfmander[filaj + 1].Item1 - Lesfmander[filaj].Item1)
* (Lesfmander[filaj + 1].Item2 * Lesfmander[filaj + 1].Item1 +
Lesfmander[filaj].Item2 * Lesfmander[filaj].Item1)) / 2;
                area_acumuladaG += areaG;

                if (Lesfmander[filaj].Item1 == 0)
                    gamma = 0;
                else
                    gamma = 1 - (area_acumuladaG / (Lesfmander[filaj].Item1 *
Lcalalfa[filaj].Item2));

                Lcalgamma.Add((areaG, area_acumuladaG, gamma));
            }

            //foreach (var MPMP in Lcalgamma)
            //  Console.WriteLine("Eje neutro (c) = {0} mm : Momento (M) = {2}
kN.m : Axial (P) = {1} kN", MPMP.Item1, Math.Round(MPMP.Item2, 7),
Math.Round(MPMP.Item3, 7));

            return (gamma, alfa, fcc, eccu, Lesfmander);
            //return Lcalgamma;


        }

        //--------------------------------------METODO DE MANDER PARA EL
CONCRETO NO CONFINADO ----------------------------------//
        public (List<(double, double, double)>, double, double, double)
MManderNoConfinado(double fc, double Ec)
        {
            double eco = 0.002;
            double fcc = fc * (-1.254 + 2.254); // fcc = fc
            double ecc = eco * (1 + 5 * (fcc / fc - 1)); // eco = ecc
```

```
            double Esec = fcc / ecc;
            double rc = Ec / (Ec - Esec);
            double ecp = 0.0064; //Def. última para el concreto no confinado
(recubrimiento)

            //Cálculo del esfuerzo medio alfa y del centroide para cualquier def
gamma.

            int Npuntos = 300;
            double xi1 = 0;
            double alfa = 0;
            double gamma = 0;
            double eci1 = 0;
            double fci1 = 0;
            double tp = 2 * eco / Npuntos;
            double areaA = 0;
            double areaG = 0;
            double area_acumuladaA = 0;
            double area_acumuladaG = 0;
            double fc2 = fcc * 2 * rc / (rc - 1 + Math.Pow(2, rc)); //Valor de
f'c para una deformación de 2eco (0.0004)

            List<(double, double, double)> Lesfmander = new List<(double,
double, double)>(); //Lista pra. alm. esf vs def de mander no confinado
            List<(double, double, double)> Lcalalfa = new List<(double, double,
double)>(); // Lista pra. alm. val. del cál. de alfa
            List<(double, double, double)> Lcalgamma = new List<(double, double,
double)>(); // Lista pra. alm. val. del cál. de gamma


            for (double i = 0; i <= Npuntos; i++)
            {
                Lesfmander.Add((eci1, fci1, xi1));
                xi1 = eci1 / ecc;
                fci1 = fcc * xi1 * rc / (rc - 1 + Math.Pow(xi1, rc));
                eci1 = eci1 + tp;
            }

            //Es posible que se deba dividir el tramo final en varios puntos,
dependiendo del método de las capas para Momento curvatura

            Lesfmander.Add((ecp, 0, 2 * eco / ecc)); //Add el últ. punto del fc
vs ec para el concreto no conf

            for (int filai = 0; filai < Npuntos; filai++)
            {
                areaA = ((Lesfmander[filai + 1].Item1 - Lesfmander[filai].Item1)
* (Lesfmander[filai].Item2 + Lesfmander[filai + 1].Item2)) / 2;
                area_acumuladaA += areaA;
                if (Lesfmander[filai].Item1 == 0)
                    alfa = 0;
                else
                    alfa = area_acumuladaA / (fcc * Lesfmander[filai].Item1);
```

```
                Lcalalfa.Add((areaA, area_acumuladaA, alfa));
            }


            for (int filaj = 0; filaj < Npuntos; filaj++)
            {

                areaG = ((Lesfmander[filaj + 1].Item1 - Lesfmander[filaj].Item1)
* (Lesfmander[filaj + 1].Item2 * Lesfmander[filaj + 1].Item1 +
Lesfmander[filaj].Item2 * Lesfmander[filaj].Item1)) / 2;
                area_acumuladaG += areaG;

                if (Lesfmander[filaj].Item1 == 0)
                    gamma = 0;
                else
                    gamma = 1 - (area_acumuladaG / (Lesfmander[filaj].Item1 *
Lcalalfa[filaj].Item2));

                Lcalgamma.Add((areaG, area_acumuladaG, gamma));
            }
            //Console.WriteLine("No Conf. Alfa {0} Gamma {1}  f'c(2eco)
={2}",alfa, gamma,fc2);




            return (Lesfmander, gamma, alfa, fc2);

        }

    }




}
```